Horizontal Scaling for an Embarrassingly Parallel Task

Lucian Carp
lc16929

December 6, 2019

The purpose of this report is to present a Cloud Nonce Discovery (CND) system that works within a Kubernetes cluster on the AWS platform. CND distributes the brute-force search for the golden nonce to a number of pods within a cluster that scales automatically when it needs resources.

1 Golden Nonce

The main component of this system is a Python script (`cnd-worker.py`) that performs the proof of work stage of the blockchain distributed ledger. This means that its main function is to search through all the possible nonces (in this case all 32-bit numbers), until it finds one that when combined with a "block" string and then hashed with the SHA256^2 algorithm results in a hash that has the number of leading zeros equal to or greater than a difficulty number D.

This operation is further optimised through concurrency by using the `multiprocessing` library to run the nonce discovery function on all the cores of a machine. The script initiates multiple processes (each representing a core of the CPU) and gives them a different range of numbers to look through (4294967296 divided by the number of cores). The first golden nonce found by any process is used as the final result and then all the other spawned processes are terminated.

As explained further down, this script is intended to be run by multiple pods within a Kubernetes cluster. Thus, this script can take environmental variables which specify the difficulty value D that will be used in the nonce discovery and information about the pod that it is running in. By using the index of the pod, it is able to determine which range of numbers to look through for the nonce, which is then further divided to all the cores of the machine.

Once a golden nonce is found, the Boto3 library (the AWS SDK for Python) is used to send a SQS message on the `cnd` queue containing the golden nonce, time elapsed and the id of the master process, which, as explained further down, is used by the master script to identify the message.

2 Docker Image

```
FROM python:alpine3.7
COPY /cnd-worker /cnd-worker
WORKDIR /cnd-worker
RUN pip install -r requirements.txt
EXPOSE 3000
CMD python ./cnd_worker.py
```

Figure 1: *Dockerfile used to create the worker Docker Image*

The worker script, the one that performs the nonce discovery, is used to build a Docker Image which uses Alpine Linux as the base image. Alpine was chosen because it is a lightweight Linux distribution, which only has 5 MB in size. The instructions found in the Dockerfile used to build the image can be observed in Figure 1. As it can be seen, the prerequisites for the worker script are installed using a `requirements.txt` file. Once built, the image is then deployed to a Dockerhub repository called luciancarp/cnd-worker [5] with the tag `latest`.

3 AWS and Kubernetes

3.1 Overall Architecture

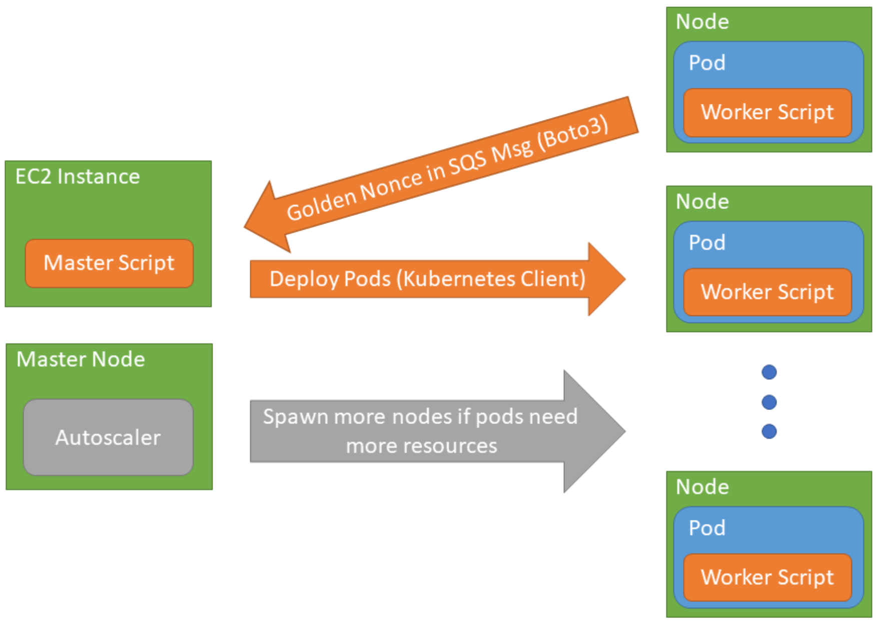
EC2 Instance
Master Script
Master Node
Autoscaler
Golden Nonce in SQS Msg (Boto3)
Deploy Pods (Kubernetes Client)
Spawn more nodes if pods need more resources
Node
Pod
Worker Script
Node
Pod
Worker Script
Node
Pod
Worker Script

Figure 2: *Architecture of the CND*

A Kubernetes cluster which uses an autoscaler was deployed on AWS using an Ubuntu EC2 instance called `k8s-management-server`. The git repository of the CND [4] was then cloned on this instance.

The repository contains a Python master script that can take as arguments a difficulty D and the number of pods that it creates when run. When it is executed, it creates multiple pods which use the `luciancarp/cnd-worker` Docker image, each being given as environmental variables the difficulty D, the process id (uuid4 created when the script is run), the index and total number of pods created for the job. Then it checks a SQS queue for a message from the pods containing the golden nonce. This is done until it either finds a nonce or it runs out of time. When it finds a message with a golden nonce on the SQS queue, it terminates all the pods and then displays the results.

3.2 Deployment

The Kubernetes cluster was deployed [8] using `kops` (Kubernetes Operations), which is a tool that "helps you create, destroy, upgrade and maintain production-grade, highly available, Kubernetes clusters from the command line" [6]. Alongside `kops`, `kubectl` was used to manage the pods and get information about their status on the command line during development. These tools are used from within an EC2 instance (k8s-management-server). To note is that a S3 bucket was created and used to store the state of the cluster. The autoscaler was deployed using a bash script given on the official `kops` Github repository [2].

In order to stay within the AWS Free Tier, the cluster is configured to only use `t2.micro` machines for the nodes, with a maximum of 20 nodes.

Deployment instructions adapted from [8] can be found in the README file of the CND [4].

3.3 Master Script

As stated above, the master script creates pods that perform the nonce discovery operation, each on a different range of possible nonces. The pods are deployed using the Kubernetes client library for Python [7], which takes in a pod manifest in a JSON format as a pod template. In this manifest are specified which container image to use, the environmental variables such as the D value or process id and also the resources that the pod will require to run. If a pod fails due to not having the the resources specified in the manifest, the autoscaler will spawn more nodes to satisfy the requirements. If the need for resources has already been met and there are unused nodes in the cluster, the autoscaler will automatically terminate the idle nodes.

After the deployment of the pods, the master script then verifies to see if any of them have found the golden nonce using the Boto3 library. Every 2 seconds, the script loads the messages from the SQS queue called `cnd`. If it finds one that has the same process id as the one given to the pods as environmental variables, then it gets the nonce and time elapsed from the message, deletes it and then stops looking into the queue. Once it receive the nonce, the script deletes all the pods created by it, again using the `kubernetes` library, then prints the results (Figure 3).

3.4 Running CND

The CND can be started by connecting to the `k8s-management-server` EC2 instance through SSH and then running the `cnd-master.py` script with the D value and number of pods as arguments. A bash script that can start the CND from a local terminal has been given in the `git` repository of the CND (it requires the private key file that was used to launch the EC2 instance). One thing to note is that, if the script is run with a large number of pods, the first execution will take longer, as it takes a few seconds for the Autoscaler to spawn new nodes for the pods.

```
2019-12-05 20:41:15.628367: Begin process f6da0b84-0f26-4c0c-a7ef-c09a948a123a
2019-12-05 20:41:15.920716: Create Pod cnd-worker-pod-0xs9f8
2019-12-05 20:41:15.930415: Create Pod cnd-worker-pod-179692
2019-12-05 20:41:15.939150: Create Pod cnd-worker-pod-2r6n5z
2019-12-05 20:41:15.950761: Create Pod cnd-worker-pod-3gvq2j
2019-12-05 20:41:15.950911: Check SQS Queue: https://eu-west-1.queue.amazonaws.com/508055761642/cnd
2019-12-05 20:41:15.973143: Check SQS Queue: https://eu-west-1.queue.amazonaws.com/508055761642/cnd
2019-12-05 20:41:15.993888: Check SQS Queue: https://eu-west-1.queue.amazonaws.com/508055761642/cnd
2019-12-05 20:41:16.015279: Check SQS Queue: https://eu-west-1.queue.amazonaws.com/508055761642/cnd
2019-12-05 20:41:16.073461: Golden Nonce not found for process f6da0b84-0f26-4c0c-a7ef-c09a948a123a
2019-12-05 20:41:18.075746: Check SQS Queue: https://eu-west-1.queue.amazonaws.com/508055761642/cnd
2019-12-05 20:41:18.134432: Golden Nonce not found for process f6da0b84-0f26-4c0c-a7ef-c09a948a123a
2019-12-05 20:41:20.136744: Check SQS Queue: https://eu-west-1.queue.amazonaws.com/508055761642/cnd
2019-12-05 20:41:20.195710: Golden Nonce not found for process f6da0b84-0f26-4c0c-a7ef-c09a948a123a
2019-12-05 20:41:22.197998: Check SQS Queue: https://eu-west-1.queue.amazonaws.com/508055761642/cnd
2019-12-05 20:41:22.210168: Golden Nonce found: 1074435534
2019-12-05 20:41:22.210168: Time Elapsed: 0:00:06.259283
2019-12-05 20:41:22.218274: Delete Pod cnd-worker-pod-0xs9f8
2019-12-05 20:41:22.237321: Delete Pod cnd-worker-pod-179692
2019-12-05 20:41:22.244940: Delete Pod cnd-worker-pod-2r6n5z
2019-12-05 20:41:22.277564: Delete Pod cnd-worker-pod-3gvq2j
2019-12-05 20:41:22.300286: End process f6da0b84-0f26-4c0c-a7ef-c09a948a123a
Results: Golden Nonce: 1074435534
Results: Found by pod: cnd-worker-pod-3gvq2j
Results: Time Elapsed pod: 0:00:02.667894
Results: Time Elapsed until received result: 0:00:06.259283
```

Figure 3: *Logs of* `python cnd-master.py --d 20 --pod-count 4`

4 Discussion

4.1 Challenges

One of the first challenges that had to be overcome, in order to develop this kind of CND, was to correctly deploy a Kubernetes cluster to AWS. This is due to the fact that most deployment instructions were either old or for Amazon Elastic Kubernetes Service (EKS), which is their managed Kubernetes service. Using `kops` proved to be the best way to deploy a Kubernetes cluster on AWS, while still having access to the EC2 instances used by the cluster.

Another difficulty was spawing pods from within the Python script. Using `kubectl` on the command line for this is straight forward, but doing this operation in Python proved to be more difficult, due to the fact that the official Python client library for Kubernetes has no documentation besides some simple examples [7]. This means that completing this part of the CND required a lot of trial and error.

There was some contradicting information regarding the autoscaler, how it works, what triggers it and how to deploy it to the cluster. This is due to the fact that there are some autoscalers that work with the CPU usages of the nodes, but these are not recommended by the official documentation [3]. Instead, the Cluster Autoscaler given by Kubernetes was used.

Working with SQS proved to be troublesome, because sometimes it took the master script more than 5 minutes to receive the message with the golden nonce from the worker script in a pod. This happened even if the messages were properly processed, as described in the Boto3 documentation [1]. This required the use of a process id every time CND was run. As the SQS queue would sometimes contain late messages from previous executions of the CND, the process id is used to check that a message comes from a pod created during the same execution of the script. This is also why there are two elapsed times displayed in the results (Figure 3), one for how long it took the pod to get find the golden nonce, and one that measures time from the creation of the pods to the time of the arrival of the message with the nonce.

4.2 Performance

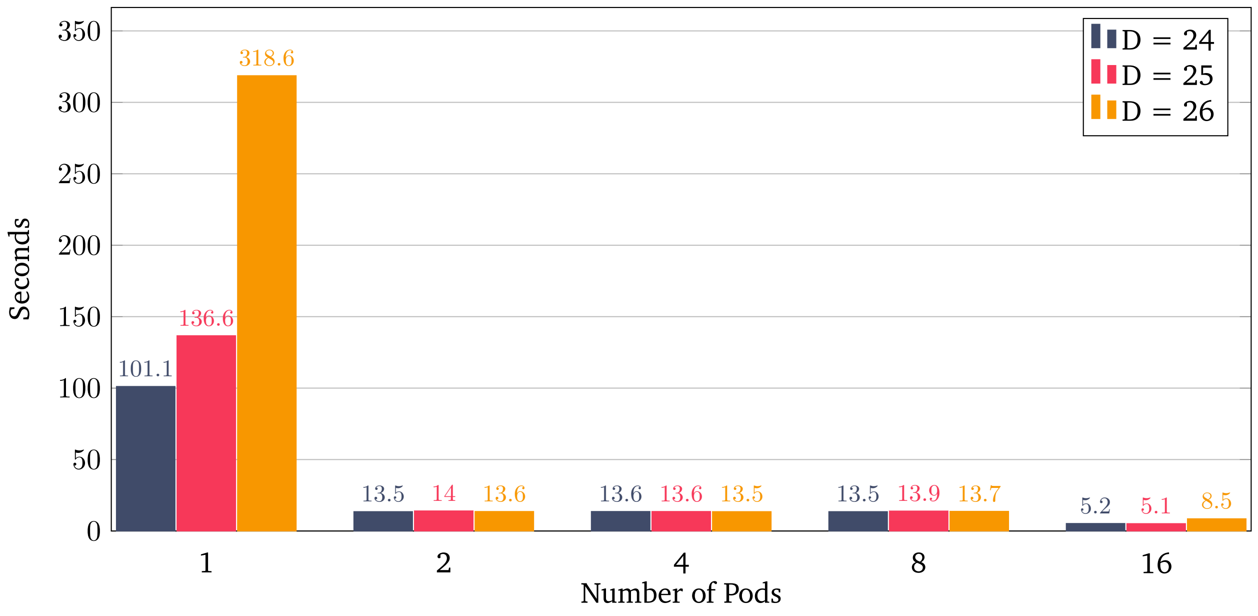

Figure 4: *The runtimes of CND*

As it can be seen in Figure 4, the runtimes become much shorter when more pods are used for the operation. For example, from 5 minutes and 18 seconds with difficulty D 26 and one pod, the runtime of CND becomes 8.5 seconds for the same difficulty when ran with 16 pods, thus decreasing the runtime by 97.3%.

4.3 Possible Improvements

One obvious improvement that could be made to improve the performance of CND is to use EC2 instances with more resources for the nodes. The `t2.micro` machines currently in use only have 1 CPU, thus the worker script cannot use concurrency to speed up the operation. Using machines with more cores would be ideal.

Another improvement would be to allow the user to indirectly specify how the CND will the divide the work and how many pods to spawn. This would be done by a taking in a desired runtime for CND and confidence level that it will find the nonce within that time. One way of doing this would be to run the CND with different difficulties and pod numbers and store the data obtained in a S3 bucket. This can be used to build a linear regression model with the pod numbers the response variable and the runtime and difficulty as covariates. A simple prediction can be used to find the number of pods for a specific desired runtime and difficulty.

5 Conclusion

In conclusion, the Cloud Nonce Discovery system presented in this report works with a Kubernetes cluster that scales automatically, depending on the resources required. Given a difficulty value and the number of pods to create, it will execute the proof of work stage of the blockchain distributed ledger, that is finding the golden nonce. By using horizontal scaling the runtime of CND can be greatly reduced.

Bibliography

[1] *Boto3 SQS Tutorial*. URL: https://boto3.amazonaws.com/v1/documentation/api/latest/guide/sqs.html.

[2] *Cluster Autoscaler Addon*. URL: https://github.com/kubernetes/kops/tree/master/addons/cluster-autoscaler.

[3] *Cluster Autoscaler Frequently Asked Questions*. URL: https://github.com/kubernetes/autoscaler/blob/master/cluster-autoscaler/FAQ.md.

[4] *CND Github Repository*. URL: https://github.com/luciancarp/cloud-computing.

[5] *Dockerhub cnd-worker Repository*. URL: https://hub.docker.com/r/luciancarp/cnd-worker.

[6] *kops - Kubernetes Operations*. URL: https://kops.sigs.k8s.io/.

[7] *Official Python client library for kubernetes*. URL: https://github.com/kubernetes-client/python.

[8] *Setup Kubernetes (K8s) Cluster on AWS*. URL: https://github.com/ValaxyTech/DevOpsDemos/blob/master/Kubernetes/k8s-setup.md.